



IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

**APPEAL FROM THE EXAMINER TO THE BOARD
OF PATENT APPEALS AND INTERFERENCES**

In re Application of: John Joseph MAZZITELLI Confirmation No.: 1932
Serial No.: 10/057,135
Filing Date: October 29, 2001
Group Art Unit: 2141
Examiner: Serrao, Ranodhi N.
Title: MULTI-THREADED SERVER ACCEPT SYSTEM AND METHOD
Docket No.: 100110992-1

MAIL STOP: APPEAL BRIEF PATENTS
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450

Dear Sir:

APPEAL BRIEF

Appellant has appealed to the Board of Patent Appeals and Interferences from the decision of the Examiner mailed May 17, 2007, finally rejecting Claims 1-6, 8-16, 18-26 and 28-30. Appellant filed a Notice of Appeal on August 17, 2007. Appellant respectfully submits herewith this Appeal Brief.

10/19/2007 EAYALEW1 00000009 082025 10057135

01 FC:1402 510.00 DA

REAL PARTY IN INTEREST

The present application was assigned to Hewlett-Packard Company as indicated by an assignment from the inventor recorded on March 12, 2002 in the Assignment Records of the United States Patent and Trademark Office at Reel 012710, Frame 0191. The present application was subsequently assigned to Hewlett-Packard Development Company, L.P. as indicated by an assignment from Hewlett-Packard Company recorded on September 26, 2003 in the Assignment Records of the United States Patent and Trademark Office at Reel 014061, Frame 0492. The real party in interest is Hewlett-Packard Development Company, LP, a limited partnership established under the laws of the State of Texas and having a principal place of business at 20555 S.H. 249 Houston, TX 77070, U.S.A. (hereinafter "HPDC"). HPDC is a Texas limited partnership and is a wholly-owned affiliate of Hewlett-Packard Company, a Delaware Corporation, headquartered in Palo Alto, CA. The general or managing partner of HPDC is HPQ Holdings, LLC.

RELATED APPEALS AND INTERFERENCES

There are no known appeals or interferences that will directly affect or be directly affected by or have a bearing on the Board's decision in this pending appeal.

STATUS OF CLAIMS

Claims 1-6, 8-16, 18-26 and 28-30 stand rejected to a final Office Action mailed May 17, 2007. Claims 7, 17 and 27 have been cancelled without prejudice or disclaimer. Claims 1-6, 8-16, 18-26 and 28-30 are presented for appeal.

STATUS OF AMENDMENTS

No amendment has been filed subsequent to the mailing of the Final Office Action.

SUMMARY OF CLAIMED SUBJECT MATTER

Embodiments of the present invention as defined by independent Claim 1 are directed toward a multi-threaded server accept method comprising: creating a socket accept thread (31b) by a control thread (31a) of a server process (31); receiving a service request from a client (40a-40n) by the socket accept thread (31b); transferring the request to a data structure (35); retrieving the request, by the control thread (31a), from the data structure (35); and transferring the request to a client thread (31c-31m) dynamically created by the control thread (31a) to

process request data associated with the request (at least at page 4, lines 5-28; page 5, lines 6-33; page 6, lines 7-27; page 8, lines 10-33; page 9, lines 15-31; and figures 1 and 2A-2C).

Embodiments of the present invention as defined by Claim 3 are directed toward the invention as defined in Claim 1, wherein the data structure (35) comprises a FIFO queue (35) (at least at page 6, lines 7-10; page 8, lines 17-19; and figures 1 and 2A).

Embodiments of the present invention as defined by independent Claim 12 are directed toward a multi-thread server accept system (10) comprising a server process (31) residing on a server (30) and operable to: create a socket accept thread (31b) by a control thread (31a) of the server process (31) residing on the server (30); receive a service request from a client (40a-40n) by the socket accept thread (31b); transfer the request to a data structure (35); retrieve the request, by the control thread (31a), from the data structure (35); and transfer the request to a client thread (31c-31m) dynamically created by the control thread (31a) to process request data associated with the request (at least at page 4, lines 5-28; page 5, lines 6-33; page 6, lines 7-27; page 8, lines 10-33; page 9, lines 15-31; and figures 1 and 2A-2C).

Embodiments of the present invention as defined by Claim 14 are directed toward the invention as defined in Claim 12, wherein the data structure (35) comprises a FIFO queue (35) (at least at page 6, lines 7-10; page 8, lines 17-19; and figures 1 and 2A).

Embodiments of the present invention as defined by independent Claim 22 are directed toward a multi-threaded server accept application (31) comprising an application software residing on a computer-readable medium and operable to: create a socket accept thread (31b) by a control thread (31a) of the application software; receive a request from a client (40a-40n) by the socket accept thread (31b); transfer the request to a data structure (35); retrieve the request, by the control thread (31a), from the data structure (35); and transfer the request to a client thread (31c-31m) dynamically created by the control thread (31a) to process request data associated with the request (at least at page 4, lines 5-28; page 5, lines 6-33; page 6, lines 7-27; page 8, lines 10-33; page 9, lines 15-31; and figures 1 and 2A-2C).

Embodiments of the present invention as defined by Claim 24 are directed toward the invention as defined in Claim 22, wherein the data structure (35) comprises a FIFO queue (35) (at least at page 6, lines 7-10; page 8, lines 17-19; and figures 1 and 2A).

GROUND OF REJECTION TO BE REVIEWED ON APPEAL¹

1. Claims 1, 2, 4-6, 8-11, 12-13, 15-16, 18-23, 25-26, and 28-30 were rejected under 35 USC §102(e) as being anticipated by U.S. Patent Publication No. 2003/0088609 issued to Guedalia et al. (hereinafter "*Guedalia*").

2. Claim 3, 14, and 24 were rejected under 35 U.S.C. §103(a) as being unpatentable over U.S. *Guedalia* as applied to respective independent Claims 1, 12, and 22 and further in view of U.S. Patent Publication no. 2001/0029548 issued to Srikantan et al. (hereinafter "*Srikantan*").

ARGUMENT

A. Standard

1. 35 U.S.C. § 102

Under 35 U.S.C. § 102, a claim is anticipated only if each and every element as set forth in the claim is found in a single prior art reference. *Verdegaal Bros. v. Union Oil Co. of California*, 2 U.S.P.Q.2d 1051 (Fed. Cir. 1987); M.P.E.P. § 2131. In addition, "[t]he identical invention must be shown in as complete detail as is contained in the . . . claims" and "[t]he elements must be arranged as required by the claim." *Richardson v. Suzuki Motor Co.*, 9 U.S.P.Q.2d 1913, 1920 (Fed. Cir. 1989); *In re Bond*, 15 U.S.P.Q.2d 1566 (Fed. Cir. 1990); M.P.E.P. § 2131.

2. 35 U.S.C. § 103

To establish a *prima facie* case of obviousness under 35 U.S.C. § 103, three basic criteria must be met: First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings; second, there must be a reasonable expectation of success; and finally, the prior art reference (or references when combined) must teach or suggest all the claim limitations. *In re Vaeck*, 947 F.2d 488, (Fed. Cir. 1991); M.P.E.P. § 2143. The teaching or suggestion to make the claimed combination and the reasonable

¹ The Appellee does not explicitly specify the ground(s) for rejecting each of Claims 12-16, 18-26, and 28-30. Instead, the Appellee merely states that Claims 12-16, 18-26, and 28-30 have limitations similar to Claims 1-6 and 8-11 and are, therefore, rejected under the same rationale. Therefore, Appellant assumes that the grounds for rejecting Claims 12-16, 18-26, and 28-30 are as indicated below.

expectation of success must both be found in the prior art, and not based on applicant's disclosure. *Id.* Further, the mere fact that references can be combined or modified does not render the resultant combination obvious unless the prior art also suggests the desirability of the combination. *In re Mills*, 916 F.2d 680 (Fed. Cir. 1990); M.P.E.P. § 2143.01. Additionally, not only must there be a suggestion to combine the functional or operational aspects of the combined references, but also the prior art is required to suggest both the combination of elements and the structure resulting from the combination. *Stiftung v. Renishaw PLC*, 945 F.2d 1173, 1183 (Fed. Cir. 1991). Moreover, where there is no apparent disadvantage present in a particular prior art reference, then generally there can be no motivation to combine the teaching of another reference with the particular prior art reference. *Winner Int'l Royalty Corp. v. Wang*, 202 F.3d 1340, 1349 (Fed. Cir. 2000).

B. Argument

1. Rejection under 35 USC §102

a. Claims 1, 2, 4-6, and 8-11

Claims 1, 2, 4-6, and 8-11 were rejected under 35 USC §102(e) as being anticipated by U.S. Patent Publication No. 2003/0088609 issued to Guedalia et al. (hereinafter "*Guedalia*"). Claim 1 is independent. Appellant respectfully submits that Claim 1 is patentable over the cited references and, therefore, Claims 2, 4-6, and 8-11 that depend respectively, therefrom, are also patentable.

Independent Claim 1 recites "transferring the request to a client thread dynamically created by the control thread to process request data associated with the request" (emphasis added). Appellant respectfully submits that *Guedalia* does not disclose or even suggest all the limitations of Claim 1. For example, *Guedalia* appears to disclose a multi-thread management system for processing requests by a server from a client. (*Guedalia*, Abstract and paragraph 0240). *Guedalia* appears to create a finite number of threads within a thread pool to process the requests. (*Guedalia*, 0244). *Guedalia* appears to use a "watchdog" thread to monitor the activity associated with each thread using a tick counter. (*Guedalia*, 0240 and 0243). In *Guedalia*, at every 50 msec interval, the "watchdog" appears to check to see if a particular thread is active, and, if so, increments the tick counter for that particular thread. (*Guedalia*, 0244). If the tick counter for a particular thread in *Guedalia* exceeds a particular threshold (e.g., 3), then the "watchdog" appears to lower the priority of the thread, remove the thread from the thread pool, and create a new thread to replace that thread in the thread pool. (*Guedalia*, paragraph 0245). Thus, Appellants respectfully submit that *Guedalia* appears to disclose

processing request data from already existing threads in a thread pool and does not appear to have threads "dynamically created by the control thread to process request data associated with the request" as recited in Claim 1 (emphasis added).

In fact, Appellant respectfully submits that *Guedalia* appears to teach away from "dynamically created" threads to "process request data associated with the request" as recited in Claim 1. For example, *Guedalia* states:

When a request queues up, the system of the present invention does not immediately create a new thread. Rather, the watchdog manages the threads. Whenever the watchdog discovers, during its regular check, that the tick counter of a thread has reached 3, it then lowers the priority of this thread and removes it from the thread pool, and creates a new thread to replace it. The old thread that was removed from the thread pool completes its task and dies.

(*Guedalia*, paragraph 0245). *Guedalia* further recites:

The curve labeled 104 in FIG. 3 corresponds to a server using a preferred embodiment of the present invention, and is approximately four times faster than the IIS server. This is because the present invention dynamically allocates threads using a "watchdog" algorithm to monitor threads rather than requests, and was able to process all of the client requests with only 2-3 threads.

(*Guedalia*, paragraph 0252). Additionally, *Guedalia* recites:

The ITS server allocated approximately 54 threads, and assigned threads to each client. These threads competed for memory pages, and as a result page faults were rampant.

(*Guedalia*, paragraph 0253). *Guedalia* also recites:

In contrast, the server of the present invention, whose performance is indicated in curve 104 waited a short time (approximately 150 msec.) before assigning threads to client requests, to see if an active thread in the thread pool would be freed up and could then be re-used for processing a queued request. Even though the client requests had to wait in a queue, the overall performance was better due to the fact that there were a smaller number of concurrent active threads. Using the IIS server, client requests did not necessarily have to wait in a queue, and were immediately assigned to a waiting thread, but the proliferation of threads caused enough page faults that the overall performance was worse.

(*Guedalia*, paragraph 0254). The cited text of *Guedalia* appears to indicate that *Guedalia* "waits" before any new threads are created in order to prevent the "proliferation of threads"

which could result in additional page faults and affect the performance of the system. (*Id.*). *Guedalia* further appears to disclose that threads are created when a thread is removed from the thread pool, and when created, is placed into the thread pool. Therefore, Appellants respectfully submit that *Guedalia* would not and does not appear to disclose or even suggest “transferring the request to a client thread dynamically created by the control thread to process request data associated with the request” as recited in Claim 1 (emphasis added). Thus, for at least these reasons, Appellant respectfully submits that Claim 1 is patentable over *Guedalia*.

Claims 2-6, 8-11, 13-16, 18-21, 23-26 and 28-30 that depend respectively from independent Claim 1 are, therefore, also patentable. Thus, Appellant respectfully submits that the rejection of Claims 1, 2, and 4-11 is improper. Accordingly, Appellant respectfully requests that Claims 1, 2, 4-6, and 8-11 be allowed.

b. Claims 12-13, 15-16, 18-21

Claims 12-13, 15-16, and 18-21 were rejected under 35 USC §102(e) as being anticipated by *Guedalia*. Appellants respectfully submit that independent Claim 12 is patentable over the cited reference and, thus, remaining Claims 13, 15-16, and 18-21, which depend from independent Claim 12, is also patentable.

Independent Claim 12 recites “a server process residing on a server and operation to...transfer the request to a client thread dynamically created by the control thread to process request data associated with the request” (emphasis added). Appellant respectfully submits that *Guedalia* does not disclose or even suggest all the limitations of Claim 12. For example, *Guedalia* appears to disclose a multi-thread management system for processing requests by a server from a client. (*Guedalia*, Abstract and paragraph 0240). *Guedalia* appears to create a finite number of threads within a thread pool to process the requests. (*Guedalia*, 0244). *Guedalia* appears to use a “watchdog” thread to monitor the activity associated with each thread using a tick counter. (*Guedalia*, 0240 and 0243). In *Guedalia*, at every 50 msec interval, the “watchdog” appears to check to see if a particular thread is active, and, if so, increments the tick counter for that particular thread. (*Guedalia*, 0244). If the tick counter for a particular thread in *Guedalia* exceeds a particular threshold (e.g., 3), then the “watchdog” appears to lower the priority of the thread, remove the thread from the thread pool, and create a new thread to replace that thread in the thread pool. (*Guedalia*, paragraph 0245). Thus, Appellants respectfully submit that *Guedalia* appears to process request data from already existing threads in a thread pool and does not appear to have threads “dynamically created by the control thread to process request data associated with the request” as recited in Claim 12 (emphasis added).

In fact, Appellant respectfully submits that *Guedalia* appears to teach away from “dynamically created” threads to “process request data associated with the request” as recited in Claim 12. For example, *Guedalia* states:

When a request queues up, the system of the present invention does not immediately create a new thread. Rather, the watchdog manages the threads. Whenever the watchdog discovers, during its regular check, that the tick counter of a thread has reached 3, it then lowers the priority of this thread and removes it from the thread pool, and creates a new thread to replace it. The old thread that was removed from the thread pool completes its task and dies.

(*Guedalia*, paragraph 0245). *Guedalia* further recites:

The curve labeled 104 in FIG. 3 corresponds to a server using a preferred embodiment of the present invention, and is approximately four times faster than the IIS server. This is because the present invention dynamically allocates threads using a “watchdog” algorithm to monitor threads rather than requests, and was able to process all of the client requests with only 2-3 threads.

(*Guedalia*, paragraph 0252). Additionally, *Guedalia* recites:

The ITS server allocated approximately 54 threads, and assigned threads to each client. These threads competed for memory pages, and as a result page faults were rampant.

(*Guedalia*, paragraph 0253). *Guedalia* also recites:

In contrast, the server of the present invention, whose performance is indicated in curve 104 waited a short time (approximately 150 msec.) before assigning threads to client requests, to see if an active thread in the thread pool would be freed up and could then be re-used for processing a queued request. Even though the client requests had to wait in a queue, the overall performance was better due to the fact that there were a smaller number of concurrent active threads. Using the IIS server, client requests did not necessarily have to wait in a queue, and were immediately assigned to a waiting thread, but the proliferation of threads caused enough page faults that the overall performance was worse.

(*Guedalia*, paragraph 0254). The cited text of *Guedalia* appears to indicate that *Guedalia* “waits” before any new threads are created in order to prevent the “proliferation of threads” which could result in additional page faults and affect the performance of the system. (*Id.*). *Guedalia* further appears to disclose that threads are created when a thread is removed from the thread pool, and when created, is placed into the thread pool. Therefore, Appellants

respectfully submit that *Guedalia* would not and does not appear to disclose or even suggest “a server process residing on a server and operation to...transfer the request to a client thread dynamically created by the control thread to process request data associated with the request” as recited in Claim 12 (emphasis added). Thus, for at least these reasons, Appellant respectfully submits that Claim 12 is patentable over *Guedalia*.

Claims 13, 15-16, and 18-21 that depend respectively from independent Claim 12 are, therefore, also patentable. Thus, Appellant respectfully submits that the rejection of Claims 12-13, 15-16, and 18-21 is improper. Accordingly, Appellant respectfully requests that Claims 12-13, 15-16, and 18-21 be allowed.

c. Claims 22-23, 25-26, and 28-30

Claims 22-23, 25-26, and 28-30 were rejected under 35 USC §102(e) as being anticipated by *Guedalia*. Appellants respectfully submit that independent Claim 22 is patentable over the cited reference and, thus, remaining Claims 23, 25-26, and 28-30, which depend from independent Claim 22, is also patentable.

Independent Claim 22 recites “an application software residing on a computer-readable medium and operable to...transfer the request to a client thread dynamically created by the control thread to process request data associated with the request” (emphasis added). Appellant respectfully submits that *Guedalia* does not disclose or even suggest all the limitations of Claim 22. For example, *Guedalia* appears to disclose a multi-thread management system for processing requests by a server from a client. (*Guedalia*, Abstract and paragraph 0240). *Guedalia* appears to create a finite number of threads within a thread pool to process the requests. (*Guedalia*, 0244). *Guedalia* appears to use a “watchdog” thread to monitor the activity associated with each thread using a tick counter. (*Guedalia*, 0240 and 0243). In *Guedalia*, at every 50 msec interval, the “watchdog” appears to check to see if a particular thread is active, and, if so, increments the tick counter for that particular thread. (*Guedalia*, 0244). If the tick counter for a particular thread in *Guedalia* exceeds a particular threshold (e.g., 3), then the “watchdog” appears to lower the priority of the thread, remove the thread from the thread pool, and create a new thread to replace that thread in the thread pool. (*Guedalia*, paragraph 0245). Thus, Appellants respectfully submit that *Guedalia* appears to process request data from already existing threads in a thread pool and does not appear to have threads “dynamically created by the control thread to process request data associated with the request” as recited in Claim 22 (emphasis added).

In fact, Appellant respectfully submits that *Guedalia* appears to teach away from “dynamically created” threads to “process request data associated with the request” as recited in Claim 22. For example, *Guedalia* states:

When a request queues up, the system of the present invention does not immediately create a new thread. Rather, the watchdog manages the threads. Whenever the watchdog discovers, during its regular check, that the tick counter of a thread has reached 3, it then lowers the priority of this thread and removes it from the thread pool, and creates a new thread to replace it. The old thread that was removed from the thread pool completes its task and dies.

(*Guedalia*, paragraph 0245). *Guedalia* further recites:

The curve labeled 104 in FIG. 3 corresponds to a server using a preferred embodiment of the present invention, and is approximately four times faster than the IIS server. This is because the present invention dynamically allocates threads using a “watchdog” algorithm to monitor threads rather than requests, and was able to process all of the client requests with only 2-3 threads.

(*Guedalia*, paragraph 0252). Additionally, *Guedalia* recites:

The ITS server allocated approximately 54 threads, and assigned threads to each client. These threads competed for memory pages, and as a result page faults were rampant.

(*Guedalia*, paragraph 0253). *Guedalia* also recites:

In contrast, the server of the present invention, whose performance is indicated in curve 104 waited a short time (approximately 150 msec.) before assigning threads to client requests, to see if an active thread in the thread pool would be freed up and could then be re-used for processing a queued request. Even though the client requests had to wait in a queue, the overall performance was better due to the fact that there were a smaller number of concurrent active threads. Using the IIS server, client requests did not necessarily have to wait in a queue, and were immediately assigned to a waiting thread, but the proliferation of threads caused enough page faults that the overall performance was worse.

(*Guedalia*, paragraph 0254). The cited text of *Guedalia* appears to indicate that *Guedalia* “waits” before any new threads are created in order to prevent the “proliferation of threads” which could result in additional page faults and affect the performance of the system. (*Id.*). *Guedalia* further appears to disclose that threads are created when a thread is removed from the thread pool, and when created, is placed into the thread pool. Therefore, Appellants

respectfully submit that *Guedalia* would not and does not appear to disclose or even suggest “an application software residing on a computer-readable medium and operable to...transfer the request to a client thread dynamically created by the control thread to process request data associated with the request” as recited in Claim 22 (emphasis added). Thus, for at least these reasons, Appellant respectfully submits that Claim 22 is patentable over *Guedalia*.

Claims 23, 25-26, and 28-30 that depend respectively from independent Claim 22 are, therefore, also patentable. Thus, Appellant respectfully submits that the rejection of Claims 22-23, 25-26, and 28-30 is improper. Accordingly, Appellant respectfully requests that Claims 22-23, 25-26, and 28-30 be allowed.

2. Rejection under 35 USC §103

a. Claims 3, 14, and 24

Claims 3, 14, and 24 were rejected under 35 U.S.C. §103(a) as being unpatentable over *Guedalia* as applied to respective independent Claims 1, 12, and 22 and further in view of U.S. Patent Publication no. 2001/0029548 issued to Srikantan et al. (hereinafter “*Srikantan*”). Appellants respectfully submit that Claim 3, 14, and 24 are patentable over *Guedalia* in view of *Srikantan* and are therefore allowable.

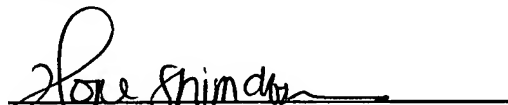
Claims 3, 14, and 24 depend from respective independent Claims 1, 12 and 22. Appellant repeats and incorporates herein the arguments presented above in connection with independent Claims 1, 14, and 24 such that *Guedalia* does not disclose or even suggest all the limitations of Claims 1, 14, and 24 and, therefore, *Guedalia* does not disclose or even suggest all the limitations of Claim 3, 14, and 24 which depend from respective Claims 1, 14, and 24. Further, the Appellee does not rely on *Srikantan* to remedy, nor does *Srikantan* appear to remedy, at least the deficiencies of *Guedalia* indicated above. Therefore, for at least this reasons, Claims 3, 14, and 24 are patentable over *Guedalia* in view of *Srinkantan*.

CONCLUSION

Appellant has demonstrated that the present invention as claimed is clearly distinguishable over the art cited of record. Therefore, Appellant respectfully requests the Board of Patent Appeals and Interferences to reverse the final rejection of the Examiner and instruct the Examiner to issue a notice of allowance of all claims.

Pursuant to MPEP § 1204.01, the previously paid appeal fee of \$500 should be applied to the filing of the instant appeal brief. Therefore, a fee of \$10.00 is believed due for the filing of the instant appeal brief. Although no other fee is believed due, the Commissioner is hereby authorized to charge any fees or credit any overpayments to Deposit Account No. 08-2025 of Hewlett-Packard Company.

Respectfully submitted,

A handwritten signature in black ink, appearing to read "Hope Shimabuku", is written over a horizontal line.

Hope Shimabuku
Registration No. 57,072

Date: October 15, 2007

Correspondence To:

Hewlett-Packard Company
Intellectual Property Administration
P.O. Box 272400
Fort Collins, Colorado 80527-2400
Tel. (970) 898-7917

CLAIMS APPENDIX

1. A multi-threaded server accept method, comprising:
creating a socket accept thread by a control thread of a server process;
receiving a service request from a client by the socket accept thread;
transferring the request to a data structure;
retrieving the request, by the control thread, from the data structure; and
transferring the request to a client thread dynamically created by the control thread to process request data associated with the request.
2. The method of claim 1, wherein the data structure comprises a queue.
3. The method of claim 1, wherein the data structure comprises a FIFO queue.
4. The method of claim 1, further comprising waiting for service requests by performing an accept () call.
5. The method of claim 1, wherein receiving the request comprises receiving a client socket object.
6. The method of claim 1, further comprising waiting for the service request from the client by the socket accept thread.
8. The method of claim 1, further comprising:
receiving a second request by the socket accept thread from the client;
transferring the second request to the data structure;
retrieving the second request by the control thread;
transferring the second request to a second client thread to process second request data; and
processing the second request data by the second client thread.

9. The method of claim 8, further comprising creating the second client thread to process the second request data.

10. The method of claim 1, wherein the socket accept thread and the control thread are executed on a single processor.

11. The method of claim 1, wherein the steps of transferring the request to the data structure and retrieving the request from the data structure are serially performed.

12. A multi-thread server accept system, comprising:

a server process residing on a server and operable to

create a socket accept thread by a control thread of a server process residing on the server;

receive a service request from a client by the socket accept thread;

transfer the request to a data structure;

retrieve the request, by the control thread, from the data structure;

transfer the request to a client thread dynamically created by the control thread to process request data associated with the request.

13. The system of claim 12, wherein the data structure comprises a queue.

14. The system of claim 12, wherein the data structure comprises a FIFO queue.

15. The system of claim 12, wherein the socket accept thread is operable to wait for service requests by performing an accept() call.

16. The system of claim 12, wherein the socket accept thread is operable to receive the request by receiving a client socket object from the client.

18. The system of claim 12, wherein the server process is further operable to:

- receive a second request from the client by the socket accept thread after transferring the request to the data structure;
- transfer the second request to the data structure;
- retrieve the second request by the control thread;
- transfer the second request to a second client thread to process the second request data; and
- process the second request data by the second client thread.

19. The system of claim 18, wherein the server process is further operable to create the second client thread to process the second request data.

20. The system of claim 12, wherein the socket accept thread and the control thread are executed on a single processor.

21. The system of claim 12, wherein the server process is further operable to serially perform the steps of transferring the request to the data structure and retrieving the request from the data structure.

22. A multi-threaded server accept application, comprising:

an application software residing on a computer-readable medium and operable to:

- create a socket accept thread by a control thread of the application software;
- receive a request from a client by the socket accept thread;
- transfer the request to a data structure;
- retrieve the request, by the control thread, from the data structure;
- transfer the request to a client thread dynamically created by the control thread to process request data associated with the request.

23. The application of claim 22, wherein the data structure comprises a queue.

24. The application of claim 22, wherein the data structure comprises a FIFO queue.

25. The application of claim 22, wherein the application software is further operable to wait for service requests by calling an accept() program.

26. The application of claim 22, wherein the application is further operable to receive the request by receiving a client socket object from the client.

28. The application of claim 22, wherein the application software is further operable to:
receive a second request from the client by the socket accept thread after transferring the request to the data structure;

transfer the second request to the data structure;

retrieve the second request by the control thread;

transfer the second request to a second client thread to process second request data;

and

process the second request data by the second client thread.

29. The application of claim 22, wherein the socket accept thread and the control thread are executed on a single processor.

30. The application of claim 22, wherein the application software is further operable to serially perform the steps of transferring the request to the data structure and retrieving the request from the data structure.

EVIDENCE APPENDIX

None

RELATED PROCEEDINGS APPENDIX

None